IntraWeb

Documentation


IntraWeb
Introduction

What is IntraWeb?

IntraWeb is a revolutionary new way to create web-based applications. Built upon Web Solution Builder and earlier versions of IntraWeb, IntraWeb extends the technology of both of these, providing an excellent tool for creating Internet, Intranet and Extranet applications in a quick and easy to maintain manner.

Many web-based development tools require the user to have knowledge of CGI scripting, state tracking and complex client configurations. IntraWeb does away with all these hassles and overheads. By simply creating the application using the component suite within Delphi, and later registering it on the server, clients can access the application using any browser that complies with HTML version 4. These include the latest versions of Netscape and Internet Explorer. Both of these have been fully tested with IntraWeb and are 100% compatible. No HTML, CGI or JavaScript coding is required; all the coding is done using Delphi. For further flexibility, the application can also be run as a stand-alone executable like any other Desktop application, providing debugging capabilities.

Documentation Sources

Be sure to check the IntraWeb FAQ (available on the Atozed Software website) as well as the information available on the website itself. A lot of documentation is contained there that is not in the manual or the help file and to keep it accurate and current we have not duplicated it.

This document is designed to be a manual, not a reference guide. The help file should be consulted when a reference for properties and components is needed.

How IntraWeb Works

IntraWeb works much like a normal executable application, with the exception that the user interface is a web browser instead of a window. After placing the application on a web server, a user can run an instance of the application by using a URL to start a session. The user's information will be tracked by the instance of the application in use, thus preventing loss of the information or mixing it up with other user information. For each user, new session information is created and tracked automatically and transparently to the developer. The overhead is low and the capacity of an IntraWeb application is similar to that of other web solutions such as ISAPI, CGI, or ASP.

IntraWeb is designed to build any sort of web-based application, whether it is a simple data entry form, a poll, or a complex application where clients have to be "logged in" for an extended period of time.

Requirements

The only requirement is for users of IntraWeb developed applications is that the browser be HTML 4 compatible, since extensive use of HTML 4 and JavaScript are made. IntraWeb has been extensively tested with Netscape and Internet Explorer and is supported with Mozilla, Netscape 6 and higher and Internet Explorer 4.0 and higher.

NOTE: If you want Netscape 4 support you should use IntraWeb version 4. Which we will continue to maintain and support.

HTML 4

IntraWeb uses HTML 4 and style sheets to achieve the coordinate placement of items and other features. Usage of templates or page mode can eliminate the need for style sheets.

JavaScript

JavaScript is used to allow many advanced client features. JavaScript also allows IntraWeb to control the browser and rendered pages. JavaScript is only required for Application Mode.

Browser Specific Features

Even with HTML and JavaScript standards in place, the browsers differ in many areas. IntraWeb adjusts for these differences automatically. IntraWeb generates the appropriate HTML and JavaScript code for the browser. IntraWeb even knows about certain bugs in specific versions of each browser and works around them dynamically. In other cases, output for each browser is optimized. See the section on Browser Implementations for more information.

Browser Implementations

Even with HTML and JavaScript standards in place, the browsers differ in many areas. Much of this is because browsers often make extensions before such features are adopted as standards. Often different browsers implement similar features in incompatible ways.

IntraWeb adjusts for these differences automatically. IntraWeb generates the appropriate HTML and JavaScript code for the browser. IntraWeb even knows about certain bugs in specific versions of each browser and works around them dynamically. In other cases, output for each browser is optimized. IntraWeb performs all of this transparently to you and without using Java, ActiveX, or any plug-ins.

Internet Explorer

Internet Explorer versions 4, 5, and 6 are supported.

Netscape 4

Users wishing to support Netscape 4 should use IntraWeb 4 which supports Netscape 4 and is the Netscape 4 version. IntraWeb 4 is still supported and maintained.

Netscape 6

Netscape 6 is supported. There are some known issues with Netscape 6 which are fixed in the recent GECKO engine used in Mozilla 1.0 RC1.

Mozilla

Mozilla RC1 is supported. Please note however that Mozilla is not officially released yet and support may change depending on bugs in Mozilla or feature changes in new releases.

Limitations of Evaluation Edition

We have designed the limitations of the evaluation edition to be such that you can fully evaluate your application with no time limits and no development restrictions. The only restrictions in the evaluation edition exist to hinder deployment.

Unique Port Numbers

When using the evaluation version of IntraWeb, the port numbers that your IntraWeb application listens will be unique upon each execution. Any port settings you may specify will be ignored. Testing is facilitated in evaluation mode by use of the execute menu item as it automatically adjusts its URL for the changing port.

IP Restriction

In the evaluation version, IntraWeb applications only listen on the IP 127.0.0.1. No requests from other IP addresses will be answered.

No Services

IntraWeb applications cannot be installed or run as services in the evaluation version. Attempts to do so will result in errors.

No SSL

SSL is disabled in the evaluation version.

No Deployment License

You may not deploy any applications created with the evaluation version.

Technical Support

Up to date support information is available at http://www.atozedsoftware.com/

Installation

Installing IntraWeb for Windows

Install

The installation will automatically integrate IntraWeb into Delphi. Three new tabs will be created on the component palette containing the IntraWeb components. One of them contains the non-database components, another contains the data-aware ones, and the third contains control components. A new tab will also be created in the Delphi repository. All IntraWeb applications should be created using the templates contained in the repository under the tab IntraWeb.

The installation process copies the appropriate files to the Windows\System directory and to sub folders of all the Delphi environments selected. It also creates program group, which can be accessed via the Start menu. The documentation is placed in this program group.

Uninstall

Uninstalling IntraWeb is done in the same way as with other Windows applications. Select Add/Remove Program from the Control Panel and click on IntraWeb to remove it from the system.

Installing License Keys

The download for registered users with license keys is the same as the evaluation edition. If you have the evaluation edition already installed you merely need to enter your license key using the registration utility. The registration utility can be run from its icon in the IntraWeb program group.

Installing IntraWeb for Linux

Development

Rethinking the User Interface

Many people try to design their web applications exactly like normal applications. If you try this, you will create interfaces that do not work well. Imagine making a windows application behave like a DOS application (WordPerfect did this with their initial Windows port). That would be an awkward interface would it not? Not only do you need to think differently about your user interface for the web, you also need to realize that the web has limitations and design around them.

One example of this is DBGrids. In a normal Delphi application, it might be considered normal to display hundreds or thousands of records in a grid. Doing such on the web will create very large HTML documents and very slow load times for the user.

Once developers realize this fact, they often ask for “Next” and “Previous” buttons and that the DBGrid be expanded to allow partial display. While this could be implemented, it would need to be implemented either to consume large amounts of memory on the server, or by constantly requiring the database which would consume less memory but would be slow. Instead of approaching it like a normal Delphi application, rethink your interface for the web.

Certainly not the only possibility, but a common one is the following technique. Instead of presenting your users with thousands of records initially, present them with a blank grid and a search field. Require your users to present some basic criteria to locate the records that they need. Using the search criteria, you can then return dozens, or just a few hundred rows. Not only is this good for bandwidth, but it is a good user interface, and will minimize the load on your database.

Allowing users to enter search criteria still allows for the possibility that the results may still number in the thousands and cause the very same problem that you were trying to avoid. To assist with this, TIWDBGrid has a RowLimit property. It defaults to 0, which means it is disabled. You can set it to a maximum value, and no matter how may rows the query returns, no more than the number in RowLimit will be returned to the user.

If you think about this, you have probably seen this technique elsewhere. Many search engines limit the number of rows that are returned. This is not only for bandwidth reasons. In most cases, the data becomes diminishingly useful after a certain number. In cases where this is not true, simply too much data is given to the user at one time and they will likely filter it anyways.

If you still decide that you do want a “paged grid” consisting of small sets of data with next / previous options you can accomplish this by setting the TIWDBGrid’s StartFirst property to false and setting the RowLimit property to the number of rows you wish to display at a given time. Then by positioning the dataset before display, you can move next / previous.

Creating a New Application

Writing your first application

All IntraWeb applications should be created using the IW Application on the IntraWeb tab in the repository. Click on File -> New and then choose the IntraWeb tab. Select Stand Alone Application. A dialog box will appear prompting for the Directory where the automatically created files should be placed.

This creates a framework for a new IntraWeb stand alone application. It copies the required files to the project directory and makes a template project file and a blank form. Although the project can be compiled and executed at this stage, it does not do anything. The standard debug form comes up displaying some information about IW and menu items to debug the application. Selecting the Execute menu item will launch the browser with a blank page. This is because main form does not contain any components or functionality yet.

```
program IWProject;

uses
      IWInitStandAlone,
      ServerController in ‘ServerController.pas’
      {IWServerController: TDataModule},
      IWUnit1 in ‘IWUnit1.pas’ {formMain:
TIWFormMain};
{$R *.res}
begin
      IWRun(TFormMain, TIWServerController);
end.
```

The code in the figure above displays the contents of the project file. It contains one procedure call to IWRun that runs the application.

```
procedure IWRun(AMainFormClass: TIWFormModuleClass;
  AServerControllerClass: TIWServerControllerBaseClass);
```

IWRun accepts two required parameters and one optional parameter. The parameters specify the Main Form class, the Server Controller class, and the Server Type.

As mentioned previously, the new project is the basic building block for any IntraWeb application. Like any other Delphi project, a main IntraWeb form is created and can be used as the main form of the application. To demonstrate the power and the facility of IW applications, below, a small example is shown.

1. Open up the default IWForm (IWUnit1.pas) that has been created.

2. Drop an IWButton, IWEdit and IWLabel on the form in no particular order.

3. Assign the following code to the IWButton.OnClick event:

```
procedure TIWFormModule.IWButton1Click(Sender: TObject);
begin
  IWLabel1.Caption := IWEdit1.Text;
end;
```

Once the steps are complete, compile and run the application. To test it, press the F9 key. The default browser should be launched and display the main form. Enter some text in the text box and click on the button. The output is displayed in the label.

Granted what we have shown in this example is not rocket science. But, it has been created using standard Delphi code and without any HTML. The example presented has been chosen because of its simplicity. The purpose is to demonstrate that programming IntraWeb applications is very much the same as developing any other Delphi application. The same methods, properties, events, etc can be used in the same way. However, IntraWeb is much more powerful and can be used to create fully enabled database applications and more.

For a more detailed introduction to creating new applications and detailed tutorials, please see the articles at http://www.atozedsoftware.com/.

Working with Forms

The repository contains another unit, which is the IntraWeb Form. All new IntraWeb forms should be created using File | New and choosing the IntraWeb Form. All forms used by an IntraWeb application MUST be an IntraWeb-specific form. Standard Delphi forms are not compatible with IntraWeb. A new unit and form will be created and displayed on the screen. Working with IntraWeb forms differs a little bit from working with standard Delphi forms. For instance, any form that is displayed must be done using the Show method of the form. In other words, ShowModal is not permitted or supported. For more information regarding forms, see the tutorials or demos.

Images and Graphics

IntraWeb supports graphics via the use of templates, TIWImage, TIWDBImage and TIWImageFile. There are many ways to use graphics with in IntraWeb, but these are the primary methods.

Templates

Use of graphics in templates is done by inserting the graphics directly into the HTML. Graphics may be served using the Files directory, or a standard web server.

TIWImage

TIWImage is used for dynamic images. Each time an image is requested the image is converted to a JPG. This can be rather resource intensive and thus should only be used for images that will be changed as part of an application's function.

For an example of this, please see the Dynamic Interactive Image demonstration in the Features demo.

For static images that are not generated each time, use TIWImageFile.

TIWDBImage

TIWDBImage converts images from a database field to a JPG automatically. It is used just like a normal TDBImage, it performs all the work necessary to display the image from the database field into the browser.

For an example of TIWDBImage, see either of the FishFact demos.

TIWImageFile

TIWImage file serves a file directly from a file on disk. Because it does no conversion of the image, TIWImageFile is an extremely efficient way to serve images and is much more efficient than TIWImage. If you are using images that are completely static, you should always use TIWImageFile.

TIWImageFile provides for design time support as well by displaying the image at design time. However the image is merely displayed, the image data is not stored with the form. Whenever displayed at design time the image is loaded from the file on disk.

The filename specifies a full path and filename to the image file to display at design time.

At run time, the path is ignored and only the filename is used. At run time, the image is expected to be in the files directory.

GIF Support

IntraWeb can support GIF files however the install does not install GIF support. Please see the IntraWeb FAQ for details on how to use GIF files with IntraWeb.

Miscellaneous

External Files

Files such as images and download can be accessed using relative paths located under the main application folder. Create a folder named files and place all HTML objects referenced inside it. In the HTML page you can reference the images using:

img src=”../files/image.jpg”

Be sure to use / and not \. Internet Explorer will correct for \, but other browsers will show broken images. In addition, this functionality is not limited to images and can be used for any file type.

Files accessed with the files URL are cached by the browser. If you wish to create dynamic files that should not be cached use ../filesnc/<filename> instead of ../files/<filename>. Files will still be retrieved from the same place in the files subdirectory, but the browser will be instructed not to cache them.

Other Form Properties

Be sure to look at the properties on the form as well. There are properties that allow customization of the output that are often overlooked. These properties allow control over the HTML output, and more.

Server Controller

Each application has a ServerController unit. The ServerController contains properties to affect how the application acts and behaves on a global scale. It also contains events that can be defined.

Datamodules

If you use datamodules, please see the FishfactDM demo. One thing to note, if you link your datasource properties to a datamodule at design time like FishFactDM does, your datamodules MUST be owned by the users WebApplication. This is done in FishFactDM by setting the datamodule's owner to the session data's owner, which is the WebApplication variable. If this is not done, the forms will not be read in properly and all the forms will be linked to the first and same datamodule.

Extending IntraWeb

Custom Components

All IntraWeb components are written using an open API that easily allows you to write your own components and add them to IntraWeb just as you can with Delphi. To further facilitate the writing of components the source code for all IntraWeb components in included, even in the evaluation edition of IntraWeb.

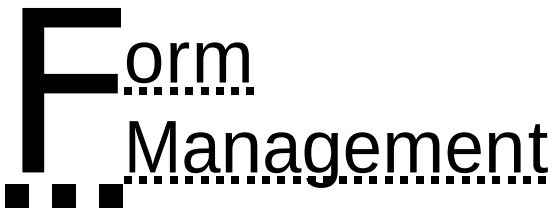
For further information on creating custom components please see the section "Writing Custom Components" in this manual.

Embedding Raw HTML

You can also embed your own HTML in your IntraWeb application without writing a component by using the TIWText component. Simply drop a TIWText component on your form. Set the RawText property to True, and the WantReturns to False. Finally, put the HTML in the Lines property and your custom HTML will be output as part of your form.

HTML Templates

HTML templates (simply referred to as templates elsewhere) can be used to add advanced HTML into your application and customize the look of your application. Please see the section on layout managers for more details.


Form
Management

Introduction

Form management in an IntraWeb application is very similar to that of a normal Delphi application but with a few restrictions.

Restrictions

1. Only one form may be visible at any time. This is because the form is actually shown in the browser.

2. Modal forms may not be used, however since only one form at a time may be visible, essentially all forms are modal.

3. Forms must be owned by WebApplication.

Usage

Form List

IntraWeb keeps a list of forms as part of the users session. It is kept in a stack like fashion with newly shown forms being added to the top. When forms are hidden or released this list is used to determine the form that should be activated if not explicitly instructed to show another form via a call to the .Show method of a form.

Normally the form list is never directly interacted with by the user but instead methods of the forms are called. However there are cases where direct interaction with the form list may be necessary. For these cases TIWApplication contains several methods for interacting with the form list and are documented in the help file.

Showing Forms

The general format to display a form is this:

```
TFormType.Create(WebApplication).Show;
```

This may be confusing at first, but it is just short hand for:

```
with TFormType.Create(WebApplication) do begin
  Show;
end;
```

This should be familiar to you as it is the same as in a standard application except the owner here is WebApplication. One thing that is different however is that in a normal application the form is shown immediately when the .Show method is called. With an IntraWeb application the call to the .Show method merely sets the form as the next active

form to show after the event exits and returns to IntraWeb. Only after the event executes will the form be shown.

A given instance of a form can be shown multiple times to bring it to the top. In this case the instance of the form will be in the form list in multiple places.

Destroying Forms

In a normal application when a form is no longer needed it can be destroyed using the .Free or the .Destroy methods.

In an IntraWeb application it is similar, however you must not call the .Free or .Destroy methods directly. Instead you must call the .Release method. The .Release method does not actually destroy the form when called. The form will not be destroyed until the event exits and returns control to IntraWeb. This is because .Release is usually called from within and event of the form itself, although this is not always the case.

After release is called, just like in a normal application the active form becomes the one that was active prior to the destroyed form became active. If you do not wish to return the user to the prior form you must call the .Show method for a different form. The .Show method can be called before or after .Release since neither takes effect until control is returned back to IntraWeb.

When a form is released, all references to it in the form list are removed. This causes an alteration in the order of the forms that will be shown when forms are hidden or released with no explicit .Show method calls.

Hiding Forms

In a normal application a form can be hidden without destroying the form by calling the .Hide method.

The same functionality can be implemented in IntraWeb by calling its simply calling the .Hide method. The .Hide method will hide the form without destroying it as .Release does.

.Hide removes all references in the form list as .Release does but does not destroy the form. Because of this you must keep a reference to the form if you wish to redisplay it later, otherwise the form will become orphaned.

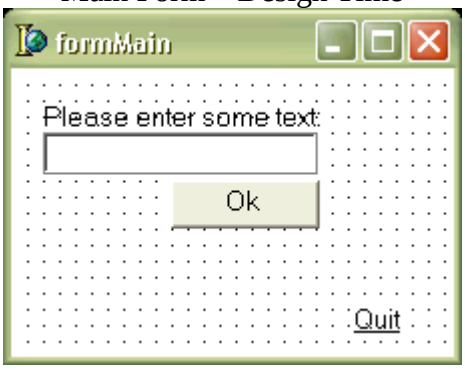
Passing Data Between Forms

Data can be passed between form just like in any normal application. Since forms are persistent information can be stored in member variables of form classes.

For demonstration purposes we will define two forms, TFormMain and TFormDialog. TFormMain is the main form and contains an button and an edit box. TFormDialog contains a memo field and a label.

When the user presses the button on the first form the text from the edit box will be added to the memo on the dialog form and the form will be displayed. The dialog form will also display how many times it has been displayed and allow the user to return to the main form.

Main Form – Design Time



Main Form – Source Code

```
unit Main;
{PUBDIST}

interface

uses
  IWAppForm, IWApplication, IWTypes, IWCompButton, IWCompEdit,
Classes,
  Controls, IWControl, IWCompLabel, Dialog, IWHTMLControls;

type
  TformMain = class(TIWAppForm)
    IWLabel1: TIWLabel;
    editText: TIWEdit;
    butnOk: TIWButton;
    IWLink1: TIWLink;
    procedure butnOkClick(Sender: TObject);
    procedure IWAppFormCreate(Sender: TObject);
    procedure IWLink1Click(Sender: TObject);
```

```
  public
    FDialogForm: TformDialog;
  end;

implementation
{$R *.dfm}

uses
  SysUtils;

procedure TformMain.butnOkClick(Sender: TObject);
var
  s: string;
begin
  s := Trim(editText.Text);
  editText.Text := '';
  if s = '' then begin
    WebApplication.ShowMessage('Please enter some text.');
  end else begin
    with FDialogForm do begin
      IWMemo1.Lines.Add(s);
      Inc(FCount);
      Show;
    end;
  end;
end;

procedure TformMain.IWAppFormCreate(Sender: TObject);
begin
  FDialogForm := TformDialog.Create(WebApplication);
end;

procedure TformMain.IWLink1Click(Sender: TObject);
begin
  WebApplication.Terminate('Good bye!');
end;

end.
```

IWLink1 OnClick

This event is hooked to the link with the caption “Quit” and simply terminates the user session when the user clicks the link.

OnCreate

The OnCreate event is called when the form is created. In this event another form is created and the reference to it is stored as a member variable of this form so it can be accessed again later.

butnOk OnClick

In the OnClick event the edit box is checked for data. If no data exists WebApplication.ShowMessage is called to display a message to the user. After the message is dismissed the main form is shown again.

If the user did enter data, using FDialogForm (which was created in this form's OnCreate) is used. Data is added to the memo, and a member variable of TFormDialog is updated. It is then displayed using the .Show method. As you can see, data is very easy to pass between forms and is the same as in a normal Delphi application.

Complete Demo

To see the project in action, please see the FormData.dpr project in the Demos directory.


State
Management

Inherent State

Standard web development tools have automatic session management, but just means that it tracks session info for you. You still have to deal with the state info between pages, or proxy information in and out of a state object. The state objects are also usually limited to strings and data must be marshaled in and out of strings, which is not feasible for complex data types.

IntraWeb has something better, and that is inherent state management. What the heck is that you say? Some new buzzword? No. Ask yourself this, How do you manage state in a normal Delphi application? What? You do not have to? EXACTLY! That is how you manage state in IntraWeb.

Restrictions

Global Variables

Global variables in general should not be used. If you want to use a global variable that is "global" yet specific to each user session you need use variables that are tied to the user session as described later.

If however you want a variable that is global among all user sessions you can and should in fact use a global variable. However as IntraWeb is a threaded environment you must take the proper steps to protect the variable from concurrent access.

ThreadVars

ThreadVars should never be used in an IntraWeb application except as temporary storage under controlled circumstances. IntraWeb is based on HTTP which is stateless. This essentially means that threads are not assigned to a specific user and a user is moved between threads between HTTP requests.

Safe Storage

Form / Datamodule Members

Since IntraWeb forms and datamodules are persistent just like in a normal Delphi application you can store information as member variables and properties. Such members should be used when the form itself needs to store data about its instance or to receive input from another form.

User Session

The user session (covered more in detail in the Session Management section of this manual) contains a .Data property that can hold a reference to an object. When you need to store user specific information you can store it in the .Data property of the session. Data accepts a TObject instance and will destroy the TObject automatically when the session is destroyed. The easiest way is to create an object and add the fields that you wish, and then create your object and store it in the session's Data property when the session is created. The Phonetics demo shows an extended example of this.

When a new IntraWeb project is created a shell user session object is created for you in the ServerController. The default ServerController looks like this:

```
unit ServerController;
{PUBDIST}

interface

uses
  SysUtils, Classes, IWServerControllerBase,
  IWApplication, IWAppForm;

type
  TIWServerController = class(TIWServerControllerBase)
    procedure IWServerControllerBaseNewSession(ASession:
TIWApplication;
      var VMainForm: TIWAppForm);
  private
  public
  end;

  TUserSession = class
  public
  end;

// Procs
  function UserSession: TUserSession;

implementation
{$R *.dfm}

uses
  IWInit;

function UserSession: TUserSession;
begin
  Result := TUserSession(RWebApplication.Data);
end;

procedure
TIWServerController.IWServerControllerBaseNewSession(
  ASession: TIWApplication; var VMainForm: TIWAppForm);
begin
```

```
  ASession.Data := TUserSession.Create;
end;

end.
```

TUserSession is an empty session object that you can add members, properties and methods to. The code to create the TUserSession for each session is also created in the OnNewSession Event.

A function named UserSession also exists for easy access to the object. So if you changed the TUserSession declaration to the following:

```
  TUserSession = class
  public
    Username: string;
    Password: string;
  end;
```

You could access these properties elsewhere in your code simply as shown here:

```
UserSession.Username := 'Joe';
LPassword := UserSession.Password;
```

If you do not need a user session you may choose to eliminate it from the code. It is not necessary and is part of the default template simply as a convenience.

The class type of TUserSession can be of any type. For projects that are generated with a datamodule the TUserSession is a descendant of TComponent and not TObject as shown here. TComponent allows the session to own components such as the datamodule and allows for easier cleanup.

Complex State and the Back Button

Many people quickly discover that when building an IntraWeb application the back button in the browser does not work. By default IntraWeb disables the back button and pressing it has no effect. Please note first that this only applies to application mode. In page mode the back button is fully functional. This limitation is because of the way that IntraWeb allows and uses complex state.

Scenario – Normal Application

Imagine a normal application designed to run on the users local computer. It has five different forms, and for some of the forms multiple instances of that form may be created with different data (such as a properties dialog displaying properties about different objects). Imagine now that at any time, without warning or notice to you the programmer, the user can go to any form in the application. But not only can they go to just any form, they can go to any form, in any prior state, even to versions of the forms which have since

been removed from memory. After they move to that form, they can even interact with it. How could such an application deal with this?

Here are a few, but certainly not all of the problems:

- Forms may rely on data in databases that no longer exists because the user deleted it.
- Forms may rely on data that has since changed, and the user would be posting old and invalid data.
- Objects that were in memory have changed, or no longer exist.

Back Buttons in non IntraWeb Systems

System not built in IntraWeb usually support back buttons. However it is because they fall into one of these categories:

- **Stateless –** They are completely stateless and reconstruct state between each page. This is usually very inefficient on the server side for weblications and puts considerably extra load on databases because data is read and written unnecessarily.
- **State Streaming** – These types stream the state into and out of each web page. This consumes bandwidth and slows down page accesses. They also cannot use complex data, or usage of complex data causes the same problems described prior.

Even applications that support the back button, such problems are still encountered. However because they allow old data to be posted they must check the data to see if the requested operations can be performed. This adds significantly to the amount of user code except in the simplest of systems. Such systems are typically not weblications, but individual dynamic pages.

IntraWeb is Not Alone

If you try many online banking applications or ordering systems, many of them have the same restrictions, but do not behave as well. Most of them allow you to go back, but will tell you that you have requested expired content. That is certainly very user unfriendly and confusing to non technical people.

Back Button for Historical Purposes

Under limited circumstances the back button can be supported in application mode. It can be enabled for historical purposes. This means that the back button will be enabled, and the user can move backward. However if they try to interact with data on a page that they

have reached using the back button they will fail. If the user tries to interact with such a page, a warning will be displayed:

```
You have attempted to post or refresh data from a page that
depends on information that is no longer available to the
server application.

Your attempted changes will be ignored. You will now be
resynchronized to the current place in the application.
```

After this warning is shown, the user will be shown the current form as it was before they used the back button.

This functionality can be turned on by setting the .HistoryEnabled property to true in the server controller.

This warning dialog can also be turned off. To do so set the .ShowResyncWarning property to false in the server controller. If false, instead of seeing the warning dialog the user will simply be resynchronized with the current form.


Session
Management

WebApplication Object

TIWApplication is to an IntraWeb application, what TApplication is to a standard Delphi application. Like the latter, TIWApplication is not a visual component. It does not appear on the property panel and therefore does not have published properties. However, it has a number of public properties and methods that can be accessed via code in the IW application. For each user session, a TIWApplication object is created. It represents the user's "application" or session.

Session Lifetime

A users session is automatically managed by IntraWeb. When a new session is started IntraWeb will create a new instance of a TIWApplication for the user and track it automatically. It can then be used to acquire information about the user, control the users session, or store additional information. No management on the developers part is required to implement session management, or to track the user.

A session exists until it is manually terminated by calling one of TIWApplication's terminate methods, or a timeout occurs. If a user does not access the application within a given time period, the user's session will be destroyed. The default timeout period is 10 minutes, but can be adjusted by changing the SessionTimeout property in the applications ServerController.

Session Implementation

Sessions are managed automatically by IntraWeb. Sessions are stored in memory on the server and there fore are secure from users who may attempt to modify the session data.

Each session is assigned a unique session ID that is used to identify the session. The session ID is constructed in a secure manner so that session IDs are not predictable and thus prone to hacking. In addition each session is tied to the users browser and if another browser is detected attempting to use the same session an error will be returned.

For further security the ServerController's RestrictIPs property. This will check the user's IP address against the session and return an error if the IP address changes. This option is false by default and should only be set to true in Intranets or Extranets with controlled clients. This is because some proxy servers such as Microsofts ISA proxy server change IP addresses between HTTP requests for a given user and will cause multiple IP addresses to be seen by the IntraWeb server.

By default the session ID is embedded in each HTML page and tracked with each HTTP request. This allows a single user to have multiple sessions per application. The disadvantage is that once the user is inside the application they cannot leave the

application and return to it. Because of this when using this method of session ID tracking any non application web pages must be opened in new windows unless it is in response to the application terminating.

Session tracking can be set to use cookies instead of embedding in the HTML page by setting the ServerController's SessionTrackingMethod property to tmCookie. This will instruct IntraWeb to use cookies to track the user's session instead. The advantage is that the user can move in and out of the application to other web pages with ease. The disadvantage is that many users disable cookies and also that the user can only have one session per application.

Accessing the Session

The users application can be accessed in several ways.

WebApplication Property of the Form

In any event or method of your forms you can simply use WebApplication which will reference the form's WebApplication property. This will meet the requirements in nearly all cases. However some notable exceptions where this property is not accessible are global procedures, TFrames, datamodules, and non IntraWeb classes.

WebApplication Property of a Control

The base IntraWeb control also contains a WebApplication property that can be used when writing custom controls.

RWebApplication

RWebApplication is a special global variable that can be used to access the current users session when a form or a control reference is not available. This is useful in datamodules, TFrame's, global procedures, and non IntraWeb classes. RWebApplication is a ThreadVar and therefore no special precautions need be taken by the developer to access it. To reference RWebApplication you will need to add IWInit to your uses clause.

Storing Additional Data

Additional data can be stored in the .Data property and is covered in the State Management section of this manual.

Session Related Events

The server controller has events related to session management that are fired for session creation and destruction.

OnNewSession

OnNewSession is fired each time a new user session is created. It passes in two arguments, ASession and VMainForm.

ASession is a reference to the newly created session and can be used to query information about the user or modified with custom information such as creating an object to be stored in the .Data property.

VMainForm is passed as a var parameter. It is initialized to nil and if not set the default main form as specified in the project file (dpr) will be used. VMainForm however can be modified based on parameters passed on the start URL, or based on other criteria to specify a main form for the user. To specify an alternate main form simply create it and return its instance in the VMainForm argument.

OnCloseSession

OnCloseSession is called when a users session is about to be terminated. This occurs either when one of the forms of WebApplication.Terminate is called, or the session has timed out.

Memory Consumption

The base memory consumption per session is quite low and in most cases is not a major consideration. The actual size can vary from session to session, but the base memory consumption excluding any user data in the .Data property should typically be at maximum 1024 bytes.

Reference

More information on the methods and properties of the TIWApplication is available in the IntraWeb help file.

Debugging

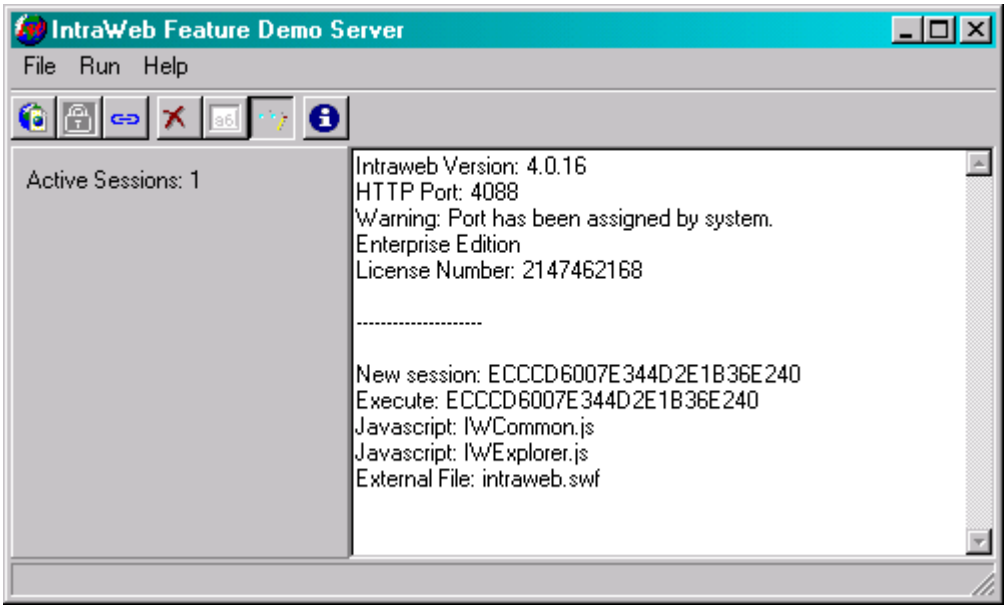
Debugging

When using standalone mode debugging is the same as any other Delphi application. Just set your break points, watches, etc and run.

Debug Output

While running in stand alone mode you can turn on debug output to see sessions created, destroyed, and HTTP requests. You can turn on debug output you can select “Show Debug Information” from the file menu, or depress the tool bar button that has an icon of the spectacles.

This is a screenshot with the debug tool bar button depressed, and debug output information from one user session:



Detecting Errors on Startup

If errors occur during start up of an application IntraWeb will terminate the application and log the error to an .err file. The applications filename with an .err extension will be used. If you are having trouble starting an application, check for an associated .err file. The .err file is a text file and can be viewed with notepad, or edln.

Errors that occur outside of the program block such as missing required packages or statically linked DLLs cannot be detected and will not be logged in the .err file.

ayout Managers and Templates

Layout Managers

What is a Layout Manager

A layout manager assembles the HTML pieces from each component into a complete HTML page for output to the browser. IntraWeb has a base layout manager, TIWLayoutMgr that can be descended from to create new layout managers. Currently IntraWeb has two layout managers TIWLayoutMgrForm and TIWTemplateProcessorHTML. In the future, there will be other layout managers to support XML and more.

Form Layout Manager – TIWLayoutMgrForm

This is the default layout manager. If no layout manager is specified, and implicit TIWLayoutMgrForm will be created and used. TIWLayoutMgrForm creates HTML pages that have the same layout and look as the designed form.

HTML Templates

Templates allow for advanced formatting and layout of individual forms. Templates also allow a web designer to design the layout of forms without using Delphi. In short, templates allow presentation and implementation to be separated. Templates are simply special HTML files.

The use of templates still requires the browser to support HTML 4 and JavaScript.

Any framed controls will be rendered without frames when templates are used. If you wish to have them frames in the template, you should frame them by using IFrame or other method in your template.

To use templates create a ‘Templates’ sub directory in your application directory and create a <FormName>.html file. Next, for the form that you wish to apply the template to:

1. Add a TIWTemplateProcessorHTML component from the IntraWeb Control tab to your form.

2. Set the form’s TemplateProcessor to the new TIWTemplateProcessorHTML component.

Most of the template functionality should be self-explanatory by looking at the examples. To see templates in action see the Phonetics Customer Profiler demo.

For each component, the template should contain a tag of the form {%Component.HTMLName%}. HTMLName in most cases is the same as the name. When the form is generated, the tags will be replaced with the component output. The use of {%%} instead of <> allows for easier editing in WSIWYG HTML editors and is compatible with all HTML editors. The {% %} tag characters are also not considered special characters and therefore are not converted to special tags. By default, a master FORM tag will surround the body to ensure proper function of all input controls and buttons. However, in some cases this can interfere with the HTML. For such cases, see the help topic for TIWTemplateProcessorHTML.MasterFormTag.

For components on a TFrame, HTMLName differs from Name. Because a TFrame is the owner of the components contained in it, components on a TFrame can have the same name as components on other TFrame instances, or as components on the form. To circumvent this components on a TFrame set their HTMLName to the frame name + component name at run time. For instance, if a component named Label1 is on a TFrame named Frame1, the components HTMLName at run time will be Frame1Label1. {%Frame1Label1%} is the tag that need to use when using components on a frame in a template.

If you wish to use the Borland style tags <#TagName#> instead of the IntraWeb style tags you can set the TagType property to ttBorland. IntraWeb type tags are easier to use with WSYWIG HTML editors.

System Templates

System templates can be used to modify the look and layout of system messages and dialogs generated by IntraWeb.

System Dialogs

There are two specific template files called IWShowMessage.html and IWException.html. These are used to provide additional formatting to ShowMessage method and for the display of uncaught exceptions. The following tags must be present:

{%textMessage%}

{%butnOk%}

The Guess demo has these two templates implemented as an example.

Note that the template for ShowMessage has no effect when smAlert or smNewWindow is passed to ShowMessage.

System Messages

System Templates support two tags: {%Content%} and {%AppName%} which can be used to display the error message. {%AppName%} is as specified in ServerController.AppName. The tag {%AppID%}. IT refers to the application ID.

IWError.html

System errors are errors that happen outside of your application and in the server portion of IntraWeb. These errors are rare and usually consist of such things as the user entering invalid requests via URLs or trying to access expired sessions. These errors can be handled by creating a template named IWError.html.

Writing Custom Components

Writing Custom Components

Under Construction

This chapter is still under construction and will be expanded in the future. Please feel free to use our newsgroups to ask any questions related to writing custom components.

Preface

All IntraWeb components are written using an open API that easily allows you to write your own components and add them to IntraWeb just as you can with Delphi. To further facilitate the writing of components the source code for all IntraWeb components in included, even in the evaluation edition of IntraWeb.

Third Parties

If you are creating components for distribution, either free of charge or for a fee, you should consider joining the IntraWeb Third Party program. Information about this program is available on the IntraWeb website. The IntraWeb Third Party program can help you with the development of your components, as well as the distribution and promotion of your components.

Source Code

Source code is included for most of IntraWeb's components. The source code can be found in the source subdirectory of the IntraWeb directory. The API is very Delphi like and for most people quite explanatory.

Core Server

In each IntraWeb application there is a core server which is responsible for serving the actual HTTP requests. This server is of based type TIWServer and is accessible by using the global GIWServer variable in the IWServer unit. The core server has methods that are of use to component writers, and is documented in the help file as TIWServer.

Deployment

Deployment of IntraWeb Applications

IntraWeb applications can be deployed as a Windows service / Linux daemon, a standalone executable, an ISAPI application, or an Apache DSO. Using page mode IntraWeb applications can be deployed by other methods as well.

Stand Alone Mode

While writing your application in Delphi, you will probably use the standalone mode to debug your application. When run as an application, a debugging screen appears with basic statistics. This screen also contains an Execute (Run | Execute or F9) option, which can be used to test execute the application in the browser. By clicking on it, the default browser will be launched with the corresponding URL to test the application.

Command Line Parameters

Auto Browser Launch

To further expedite development you can add "/LaunchBrowser" to the application parameters to have the program launch the browser automatically each time it is run. To do this in Delphi, select the Parameters menu item from the Run menu. Then enter "/LaunchBrowser" in the Parameters field. The next time you execute your application the browser will automatically be launched.

Auto Minimize

The debug screen can be told to start minimized by passing /minimize on the command line.

No Service

The same executable is used for a standalone executable and as a Windows service. To facilitate this functionality the executable queries Windows about its status as a service. This can cause problems if the executable is not run from an administrative account. To disable this querying /noservice can be passed on the command line to instruct the executable to run as a normal executable and not try to run as a service.

Service Mode

Running IntraWeb applications as a service has its benefits and disadvantages. The disadvantages are that there is no debug screen or execute menu item. The main

advantage is that the there is no requirement for logging on to the machine in order to run the application (like any other Windows service). A few steps have to be taken prior to running the application as a service. In particular, it has to be installed as such. To do so, using the windows command prompt, change to the directory where the application resides and type:

```
Application_name –install
```

This will install it and the application will appear in the Services Applet. From there, it can be configured to run automatically or manually. There is no need to activate the "Interact with Desktop" under the properties of the service, and doing so will have no effect whatsoever.

In a similar way, if the need arises to uninstall the application, it can be done by typing:

```
Application_name –uninstall
```

Before executing this command, be sure to stop your service.

Notes:

1) Only Windows NT, Windows 2000, and Windows XP support services. Windows 95, Windows 98 and Windows ME do not support services.

2) Services do not function in evaluation mode. Attempts to do so will result in errors.

ISAPI / NSAPI / Apache DSO / CGI / Win-CGI

IntraWeb applications developed using page mode can be deployed as ISAPI, Apache DSO, NSAPI, CGI, or Win-CGI.

Application mode executables may only be deployed as ISAPI, NSAPI or Apache DSO.

Launch URLs

Linking to IntraWeb Applications

Since the whole user interface is based on a web browser, calling the application is done using a URL. The URL has a very simple format.

Stand Alone Usage

Syntax: http://<server>:<port>

Example: http://www.atozedsoftware.com:4000

ISAPI Usage

Syntax: http://<server>/<script path>/<dll>

Example: http://www.atozedsoftware.com/iw/guess.dll

Apache DSO Usage

Syntax: http://<server>/<location>

Example: http://www.atozedsoftware.com/myapp

Launch URL Option

URL's are formed by specifying the host and adding the port number (if different to the default 80). You can optionally specify a start text on the URL by setting the StartCmd property in the ServerController and then adding the value of the URL For example: http://www.atozedsoftware.com:4000/launch

Sessions

Every time this URL is entered into the browser and new session is created and the user is tracked automatically throughout the whole period that the session lasts. Optionally, parameters can also be specified when calling a new instance by passing them using POST or GET.

Passing Parameters

Parameters are passed to the application using the interrogation (?) sign after the start URL. Each parameter consists of a “parameter name” and “parameter value”. Parameters are separated from each other using the ampersand (&) sign. The following examples show how to pass two parameters named “param1” and “param2” with values “value1” and “value” respectively:

http://www.atozedsoftware.com:4000?param1=value1&param2=value2 (Stand Alone)

These parameters are available in your application by accessing the RunParams property of the TIWApplication object.

In addition, prelaunch changes can be performed in the ServerController.OnNewSession event. One such use may be to read the parameters that have been passed in an offer different users different starting forms.


Performance
Evaluation

Performance Evaluation

You will likely want to test the performance of your application. Many users test the performance improperly and thus receive misleading results. When testing be aware of the following items that can negatively impact your tests.

1. When using Internet Explorer, the first page will render quickly. However, when you click on a button or a link from the first page, Internet Explorer will then load extra libraries and cause a delay. This delay is caused by Internet Explorer and not the IntraWeb application. As you move to successive pages, you will notice that this delay no longer exists.

2. When using a browser on the same machine as the server the network is forced to use the “loopback” address. The loop-back address generally provides good performance however sometimes will introduce delays into the transfer of data.

3. When using a browser on the same machine as the server, the browser, network and application all compete for CPU, disk and memory at the same time. Most browsers are quite CPU and memory intensive, and thus negatively impact the server and your results.

4. When using Netscape and running your application from Delphi, the Delphi debugger hooks and Netscape conflict. Often you will have to task switch from the browser to the application to “unstick” the local network.

5. Anytime you run your server from within Delphi, Delphi’s debugger is active. The debugger not only consumes memory and CPU, but can also slow down the execution of your application. Under normal circumstances, this is perfectly acceptable, however keep this in mind if you are testing performance.

6. The first time you execute an ISAPI based application the web server must load the DLL and this will cause for a delay.

To properly test performance, you should run your application and browser on separate machines.


Scaling IntraWeb
Applications

Scaling IntraWeb Applications

What if my application grows too big for one server to handle? Can IntraWeb scale? Sure it can.

IntraWeb can be scaled using a variety of methods. First also consider that in many applications you can handle more with less CPU because App mode is stateful. Without state, applications often spend a lot of CPU streaming state to a DB, reconstructing state, or needlessly rerunning queries against a database. That being said, there are still times you need to scale.

Add Another Tier

Use MTS, COM, SOAP, ASTA, Midas, whatever and split that piece into multiple machines. This will take processing out of the IntraWeb application and allow it to be distributed.

Beef Up You Database Server

Add more CPU power or memory to the database server. In many systems, the database is the bottleneck, and the web application spends the majority of its time waiting on the database.

Add More Memory to Application Server

Check your memory usage and make sure that your application server has the appropriate amount of memory. Virtual memory will be used if not enough physical memory is available, but this will slow down the response time and consume CPU cycles. Eliminating the use of virtual memory will increase efficiency and capacity.

Use a Multi-Processor Server

IntraWeb servers are fully threaded. Thus, IntraWeb servers will take advantage of multiple processors if present.

Distribute the IntraWeb Application Itself

If you have reached a level requiring you to scale the actual IntraWeb application, this can be accomplished as well. What will be presented here is not the only way that an IntraWeb application can be distributed, but it is the most common. It is very simple, and effective. This method can also be used in conjunction with the previously described methods.

Step 1 – Install Multiple Application Servers

Each server will need it's own IP address. For this example, we will simply refer to them as .1, .2 and .3.

Step 2 – Create a New DNS Record

Create a new DNS entry to identify the application. For this example, we will use iwapp.atozedsoftware.com.

Step 3 – Add Multiple IPs for the DNS Record

DNS allows multiple IPs to be assigned to a given record. In our example, we would assign .1, .2, .3 to iwapp.atozedsoftware.com. When multiple IPs are assigned to a single record, the DNS server will perform rotating DNS, sometimes also referred to as round robin. This means that the first request for iwapp.atozedsoftware.com will return .1, the second will return .2, the third will return .3, the fourth will return .1, and so on.

This will distribute the load across the servers. This method does not perform true load balancing, as it does not measure the load, it just distributes it. In most applications, the law of averages applies and it is quite effective. If your application is such that it creates large imbalances, you will need to use a load balancing DNS server instead.

Step 4 – Create a Redirect Entry

On each application server create a redirect entry using the primary web server's configuration, or a page that performs a redirect to that servers actual IP. When the page or virtual entry is requested, the browser will not know that it has been redirected to an IP by the DNS server, as this is part of its normal operation. However, we must make sure that subsequent requests are routed to the same application server, as IntraWeb is stateful. Note that this only applies to Application Mode, and not Page mode. This step can be skipped for Page Mode.

The virtual entry or web page merely redirects the web browser to a URL containing its individual IP instead of iwapp.atozedsoftware.com. For example if our URL is http://iwapp.atozedsoftware.com, this entry might redirect the browser to http://x.x.x.1:6000/start. This URL demonstrates a stand alone IntraWeb application, but it can be adjusted to redirect to a static page, an ISAPI version, or a DSO version. The important thing is that the browser is redirected to the physical application server so each subsequent request will return to that server.


Secure
HTTP / SSL

SSL – Secure HTTP

If your application is deployed as an ISAPI DLL or an Apache DSO, you need to use the hosting web server's SSL capabilities since it handles the HTTP protocol.

In Stand Alone mode, SSL is supported also. The first step is that you must obtain SSL certificates.

Enabling SSL

IntraWeb requires that your certificates are .pem format. To enable SSL support, follow these steps:

1. Download and install the SSL DLLs. Information on how to obtain the DLLs is available at http://www.nevrona.com/indy/ssl.html. The DLLs are free.

2. Set the SSLPort in the ServerController to a value other than 0. The default for SSL support is 443. If you are running a standard web server on the same machine and it supports SSL, it will already be using 443 and you will need to use another port.

3. Set the SSLCertificatePassword in the ServerController if you assigned a password to your certificates.

4. Place your certificates in the same directory as the application. The certificates must be named:

a. Cert.pem

b. Root.pem

c. Key.pem

Converting Certificates to PEM Format

Chances are that your certificates were not delivered to you in .pem format. If they are not in .pem you must convert them for use with IntraWeb.

This procedure assumes that you have already received your key and certificate pair from some Certificate Authority (like Verisign or Thawte) and that you have them installed in Microsoft Internet Explorer in Personal Certificates Store.

Export Certificate

Select the certificate and export it as a .pfx file (Personal Exchange Format). You may optionally protect it with a password.

Convert .pfx to .pem

As part of the SSL download, a utility named openssl.exe was included. This utility will be used to convert your .pfx file.

To use openssl.exe, use the following format:

openssl.exe pkcs12 –in <your file>.pfx –out <your file>.pem

Openssl.exe will prompt you for a password. Enter it if you used one, or leave it blank if you did not specify one. It will also prompt you for a new password for the .pem file. This is optional, but if you protect it with a password be sure to fill in the ServerController.SSLCertificatePassword property in your application.

Splitting the .pem File

If you examine the new .pem file with a notepad, you will notice that it consists of two parts. The two parts consist of the private key and the certificate (public key) part. There is also some addition information included. IntraWeb requires that this information be separated into separate files.

Key.pem

Create key.pem with notepad and paste everything between and including these two statements:

-----BEGIN RSA PRIVATE KEY-----

-----END RSA PRIVATE KEY-----

Cert.pem

Create cert.pem with notepad and paste everything between and including these two statements:

-----BEGIN CERTIFICATE-----

-----END CERTIFICATE-----

Root.pem

The final file that IntraWeb requires is the Certificate Authority certificate file. You can obtain this from the Internet Explorer in Trusted Root Certificate Authority dialog. Select the Authority that issued your certificate and export it in Base64 (cer) format. This format is also the same as PEM format so after export simply rename the file to root.pem.